北京林业大学经济管理学院

《深度学习》结课论文

论文题目	卷积神经网络的探索性学习及实战
班级	电商 18
姓名	王佳琦
学号	180412126
指导教师	翟祥

2021 年 7 月

卷积神经网络的探索性学习及实战

摘要: 本论文从加载数据集开始，通过卷积神经网络的官方教程对 cifar10 进行分类。在原代码的基础上，通过 visdom 进行可视化，并增加迭代次数，对网络搭建的一些问题进行了实验和解释。在此基础上，尝试超参数优化，提高模型的准确度。最后，运用 VGG-16 网络进行实战，实现风格迁移。

关键词: CNN　VGG　风格迁移

目录

1 加载 CIFAR10 数据集

1.1 概念及问题

首先需要弄清楚如何使用 pytorch 加载图像。用 pytorch 加载图像，需要简单了解下列几个概念。

CIFAR-10：彩色图片数据集，有 10 个类别。

Tensor：可认为是一个高维数组，可以与 numpy 对象相互转换。

Torchvision：封装数据集，包括我们要用到的 CIFAR10。

Transforms：是 pytorch 中的图像预处理包，包含了很多种对图像数据进行变换的函数。其中，Compose 方法是将多种变换组合在一起。ToTensor()将 PILImage 转变为 torch.FloatTensor 的数据形式；Normalize 则是对 tensor 进行正则化。

Dataset: 数据集，按下标访问，返回（data，label）数据。

Dataloader: 可迭代对象，对 dataset 返回的每一条数据样本拼接成一个 batch。

通过 pytorch 先加载 CIFAR10 数据集，详细代码见附录。

以下列举几个我对该段代码的理解：

1. transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))

前面的(0.5, 0.5, 0.5)是 RGB 三个通道上的均值， 后面(0.5, 0.5, 0.5)是三个通道的标准差，正则化对每个通道执行以下操作：image =（图像-均值）/标准差，转化为（-1,1）之间。

2. 在函数 imshow 中的 img = img*0.5 + 0.5

正如上一条，因为图像被正则化了，在显示图像的时候需要还原。

3. 在函数 imshow 中的 plt.imshow(np.transpose(npimg, (1, 2, 0)))

在 pytorch 中张量 tensor 对图象的存储是(图片数量，通道数，图片高，图片宽)。单独说 tensor 中的某张图片，也就是(管道数，宽，高)。而标准的 rbg 图象是(宽，高，管道数)。transpose()函数的作用就是调换数组的行列值的索引值，假设三维数组当中的索引值为 x，y，z，transpose()函数的作用就是调换 x，y，z 的位置，也就是数组的索引值。所以我们正常的数组索引值为（0，1，2），等于（x，y，z），这里用（1,2,0），就是相当于把索引变为（y，z，x）。

1.2 代码逻辑

Cifar-10 数据集是打包在 dataset 中的，所以直接调用 dataset 获取数据，同时用设定好的 transforms 对图像做一个预处理，这里包括将读取的 numpy 转化为 tensor，并且进行正则化。这里有一个点不太理解，那就是为什么要进行正则化？虽然是数学上讨论出来的，但是我决定在后文中进行实验，解答自己的疑惑。

导入数据后，定义了 imshow 函数，通过 matplotlib 显示图像，输入是(宽，高，管道数)。

2 通过简单卷积神经网络识别图像

2.1 卷积神经网络

CNN 是一种带有卷积结构的深度神经网络，通常至少有两个非线性可训练的卷积层、两个非线性的池化层和一个全连接层，一共至少 5 个隐含层[1]。

卷积层：起到滤波器的作用或者说是卷积的作用。

池化层：作用是逐渐降低数据体的空间尺寸，这样的话就能减少网络中参数的数量，使得计算资源耗费变少，也能有效控制过拟合。

全连接层：全连接层中的每个神经元与其前一层的所有神经元进行全连接. 全连接层可以整合卷积层或者池化层中具有类别区分性的局部信息.

softmax 逻辑回归: 在多分类问题中，类标签 y 可以取两个以上的值。

2.2 简单实现及代码理解

为了对卷积神经网络有个简单的了解，先对官方教程中的卷积神经网络的代码实现进行复现及分析，在此基础上，还利用了 pytorch 的可视化工具 visdom 对 loss 进行展现。

2.2.1 定义网络

Pytorch 中定义网络，需要继承 nn.Module，并实现 forward 方法，把网络中具有可学习参数的层放在构造函数__init__中。

其中，在定义层数时，

卷积层：

self.conv2d =nn.Conv2d(in_channels=3,out_channels=64,kernel_size=4,stride=2,padding=1)

前三个参数必须手动给出，如代码中运用的 nn.Conv2d(3, 6, 5)表示输入为图像（3 个频道，即 RGB 图）,输出为 6 张特征图, 卷积核为 5x5 正方形。

池化层：

self.pool = nn.MaxPool2d(2, 2)表示 kernel 为 2，步长为 2。

全连接层：

self.fc1 = nn.Linear(16 * 5 * 5, 120) 表示定义 fc1 全连接函数 1 为线性函数：y = Wx + b，并将 16*5*5 个节点连接到 120 个节点上。因为我们需要将图像分成 10 类，所以最后连接到的是 10 个节点。

在 forward 函数中，可以定义前向传播函数，pytorch 会自动利用 autograd 计算出后向传播函数。

x = self.pool(F.relu(self.conv1(x)))表示输入 x 经过卷积 conv1 之后，经过激活函数 ReLU，再进行最大池化，然后更新到 x。

x = torch.flatten(x, 1) 作用是改变张量的维度和维数。

2.2.2 定义损失函数和优化器

官方教程中使用的是交叉熵损失，以及随机梯度下降法。

2.2.3 训练网络

通过循环实现，先输入数据，前向传播，后向传播，然后更新参数。

2.3 分类结果及可视化呈现

通过 pytorch 的可视化工具 visdom，将 loss 随着迭代次数的变化记录下来。在使用 visdom 的时候，需要注意的是 loss 一般是 tensor 形式，需要转化一下。在这里，我们的 epoch 设置为了 4，即一共遍历了数据集四次。训练集的 loss 如下。

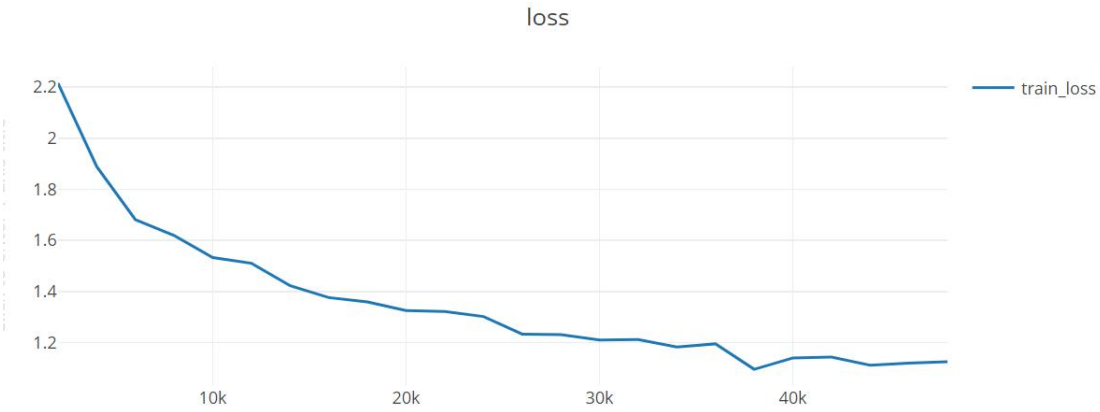


图 1 训练集的损失

在只跑 4 个 epoch 的情况下，训练集的 Accuracy 如下表所示。

表 1 迭代 4 次的结果

Accuracy for……	Accuracy（%）
All	59
Plane	70.9
Car	70.4
Bird	31.4
Cat	33.4
Deer	54.9
Dog	55.1
Frog	79.6
Horse	64.7
Ship	59.9
Truck	78.4

这里可以思考一个问题，为什么不同标签类别的准确率不同呢，这又是由什么决定的呢？我的假设是，因为卷积层提取特征不同，对不同类别的特征反映程度不同。那我们是否可以通过改善这一点，来提高类别准确度？又或者说，当我们的训练网络，只是为了去区分某一种类别的时候，可以适当牺牲其他类别的准确度。

2.4 增加迭代次数的实验结果

在官方教程当中，只显示出了 loss 和 accuracy，并且没有可视化的部分。我在官方教程的基础上，对代码稍微进行了修改，将 epoch 增加到 50，并绘制出相应的 train_loss，test_loss，test_accu，这样可以一目了然地看出网络的训练程度，以及是否有过拟合趋势。以本文的数据，没有测试集，只有训练集和验证集，下文出现的所有 test_loss 都可以等价于 val_loss 对待。

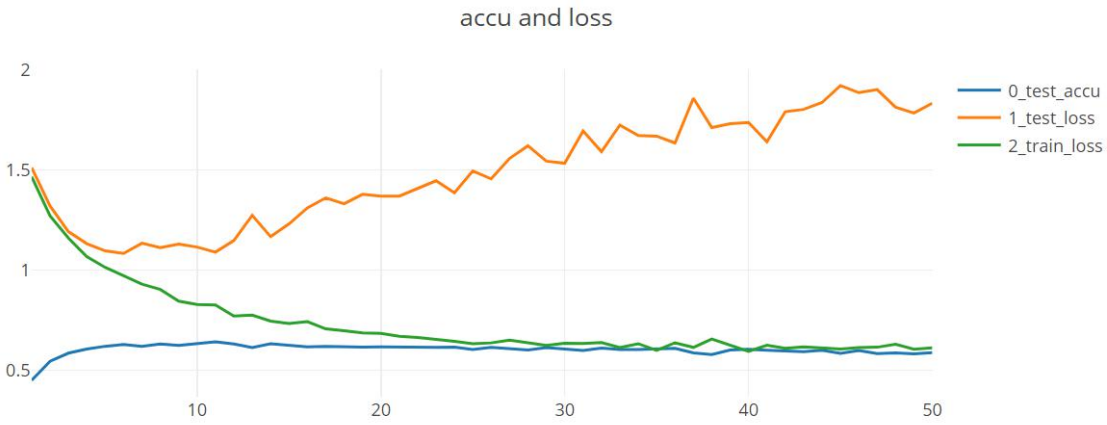


图 2 train_loss 和 test_loss

由可视化图像中可以看出，当迭代次数大于 10 时，模型在验证集上的准确度也不会随着迭代次数的增加而增加，甚至小幅度下降。而当迭代次数大于 6 次的时候，test_loss 就如脱缰的野马直线上涨，说明已经出现了过拟合。根据详细的结果，当迭代次数为 8 时，Accuracy 为 63%，而且也还没有出现过拟合。

根据 train loss 与 test loss 结果分析，train loss 趋于不变，test loss 不断上升，说明可能产生了过拟合问题。

表 2 不同迭代下的类别准确度

Accuracy for……	Epoch=4	Epoch=50	Epoch=8
All	59	57	62
Plane	70.9	54.3	63.7
Car	70.4	76.1	71.7
Bird	31.4	43.7	49.8
Cat	33.4	39	36.2
Deer	54.9	39.7	56.9
Dog	55.1	46.4	64.5
Frog	79.6	72.7	60.4
Horse	64.7	61.3	69.7
Ship	59.9	76.7	73.8
Truck	78.4	65.9	74.6

2.5 正则化对 loss 的影响

在前文中，虽然对图像进行了正则化的处理，但是并不理解这一步的用处，所以在代码中分别跑了两次，一次做了正则化处理，一次没有做，将跑出来的 loss 用 visdom 呈现，结果如下。很明显，做了正则化处理的训练集 loss 下降更快。正则化的训练集的 Accuracy 为 59%，未正则化的训练集的 Accuracy 为 50%。

图 3 正则化对损失下降的影响

在正则化的用处得到实验证实之后，回忆起老师上课讲过的内容和自己学过的知识，可以用下面的方式进行一个直观地解释。

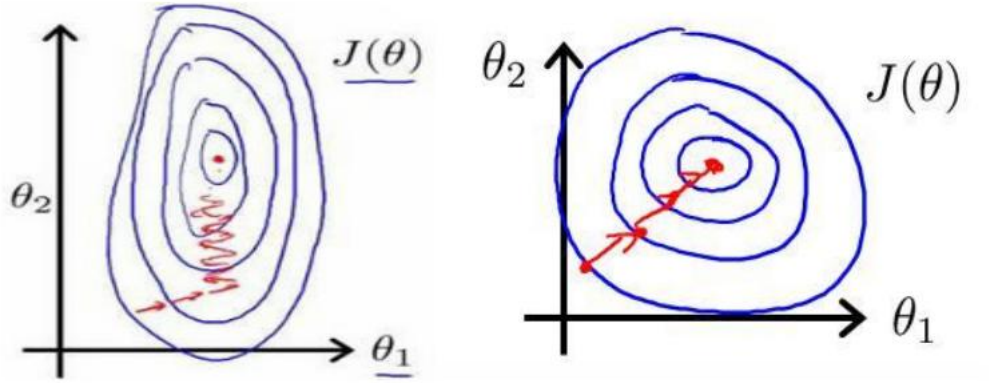


图 4 正则化前和正则化后的梯度

从图中可以看到，“细长的”等高线图就会变得“偏圆”，未经过正则化的图形，在使用梯度下降法时，会让梯度下降的过程变得不仅曲折，而且非常耗时。因此，正则化后，loss 下降得更快，也符合实验的结果。

3 基于 cifar10 的 CNN 超参数优化

3.1 学习率

好的学习率应该是有利于网络学习到参数的，或者说损失函数能够得到有效的降低。一般学习率可以从 0.1-10-8 或者更大的范围去搜索，可以每次缩小十倍。

在之前的代码中，我们只简单地使用了随机梯度下降的使用动量优化器。optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9) 表示。

更改 lr，用 visdom 呈现，结果如下。

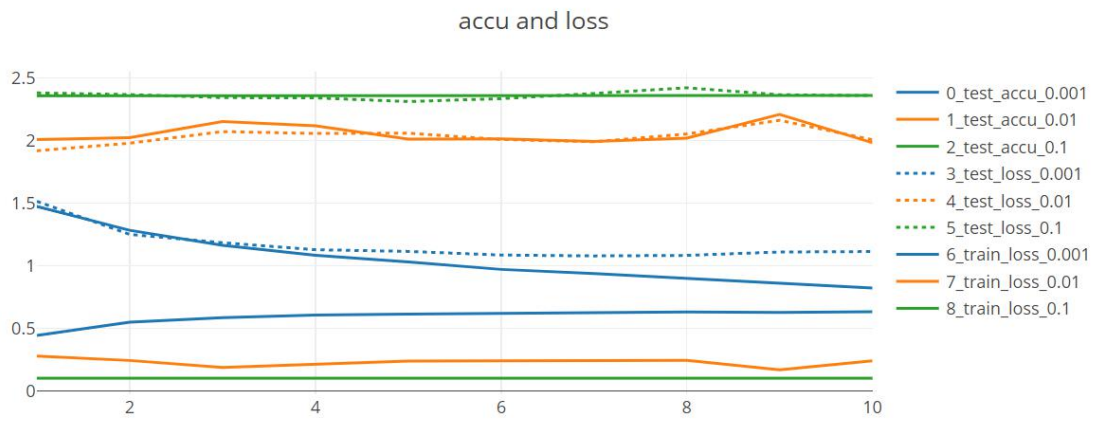


图 5 在 lr=0.1、0.01、0.001 情况下的 train_loss、test_loss、test_accu

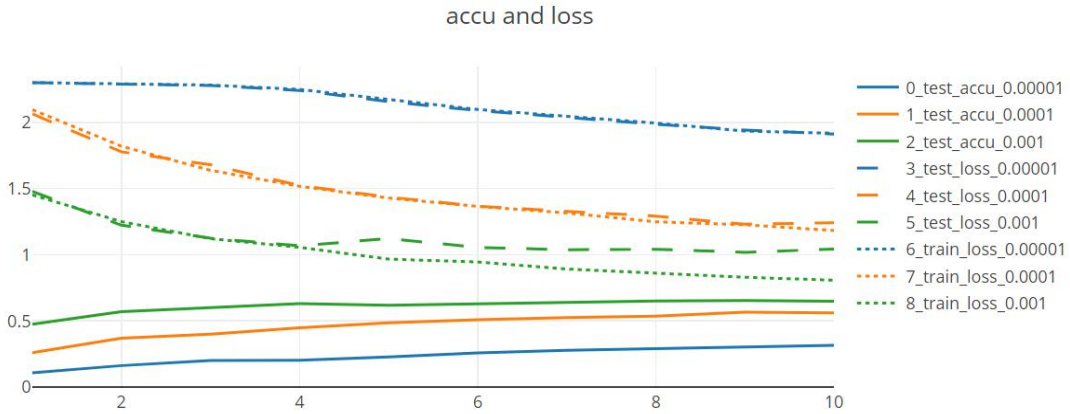


图 6 在 lr=0.001、0.0001、0.00001 情况下的 train_loss、test_loss、test_accu

上图中可以看出，当 lr 过大时，loss 并没有下降的趋势，accu 也比较低。而当 lr 过小时，准确度提高得较慢。所以将 lr 设置为 0.001 是比较合理的。

3.2 迭代次数

迭代次数是指整个训练集输入到神经网络进行训练的次数。当测试错误率和训练错误率相差较小时，可认为当前的迭代次数是合适的，否则需继续增大迭代次数，或调整网络结构。

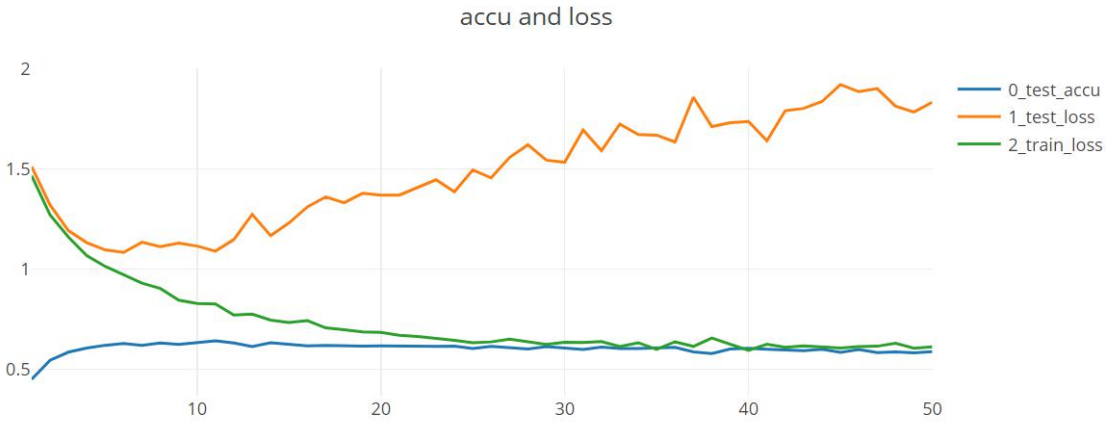


图 7 不同迭代次数下的 train_loss 和 test_loss

当迭代次数大于 10 时，模型在验证集上的准确度也不会随着迭代次数的增加而增加，甚至小幅度下降。而当迭代次数大于 6 次的时候，已经出现了过拟合。

但是，如果调整别的参数，最优的迭代次数是否会变化？

3.3 激活函数

在我们的原始神经网络中，全部采用的是 relu 激活函数，实际上不同层之间的激活函数可以不同，这样就为我们提供了多种选择。在这里，考虑以下几种激活函数。

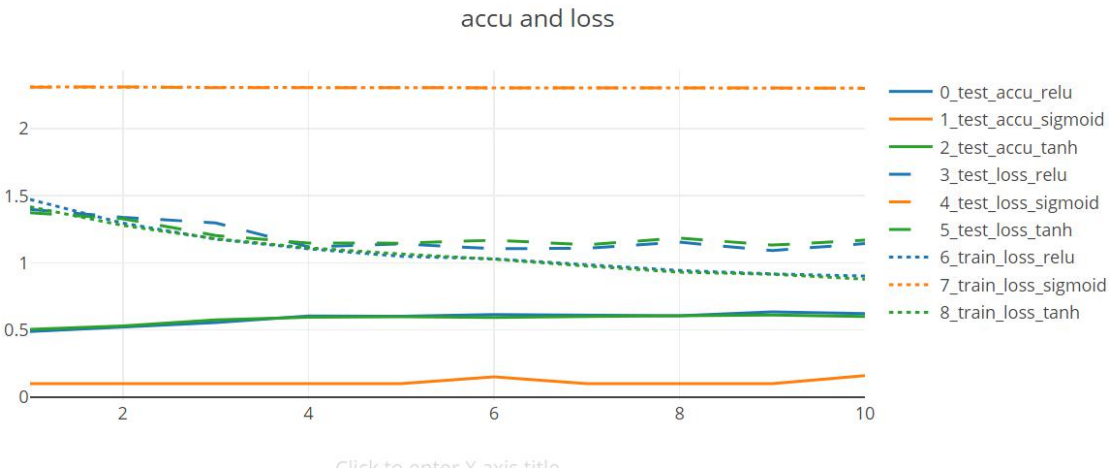


图 8 在激活函数为 relu、sigmoid、tanh 情况下的 train_loss、test_loss、test_accu

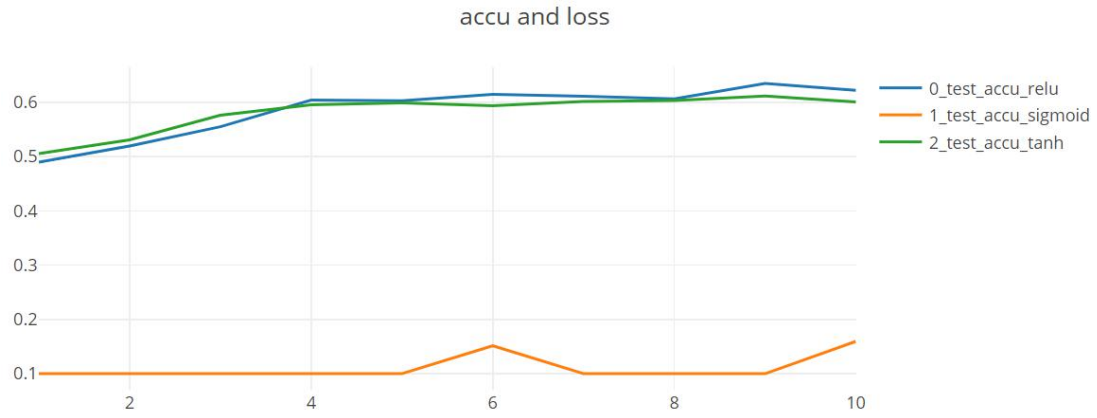


图 9 在激活函数为 relu、sigmoid、tanh 情况下的 test_accu

由上图可以看出，激活函数为 relu 时，准确度最高，而 relu 也有多种变体，在这里，尝试比较了几种，结果如图 10 所示。

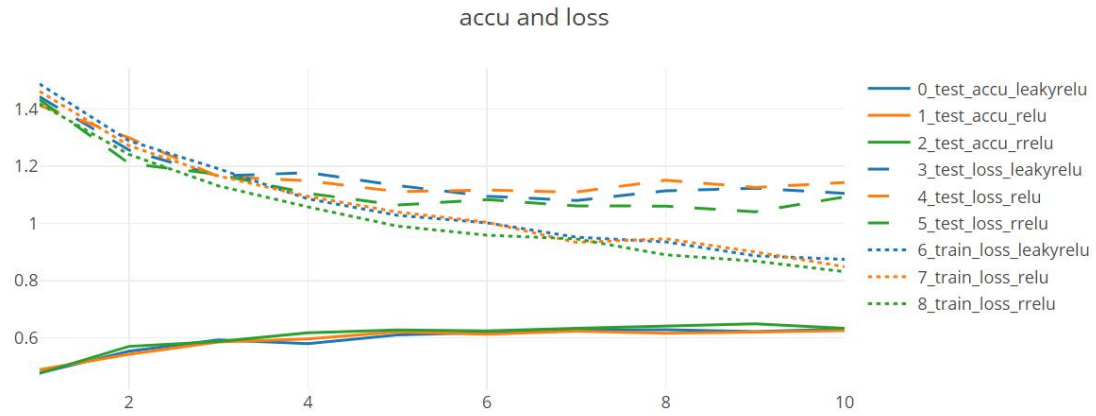


图 10 在激活函数为 leaky_relu、relu、rrelu 情况下的 train_loss、test_loss、test_accu

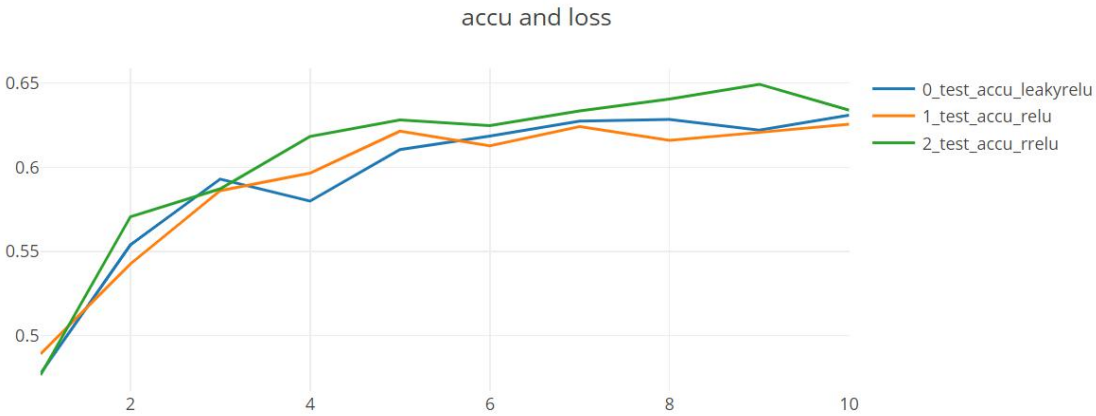


图 11 在激活函数为 leaky_relu、relu、rrelu 情况下的 test_accu

由图 11 可以看出，当激活函数为 rrelu 的时候，模型的准确度被提高到 65%。

4 进阶：使用 VGG 实现风格迁移

在上文中通过一些例子和调参尝试，了解了简单的卷积神经网络的工作方式和 pytorch 的用法，接下来使用 VGG-16 实现图像的风格迁移。

4.1 VGG-16 神经网络

VGG16 网络由 13 层卷积层+3 层全连接层组成。结构如下。VGG 网络在风格迁移中使用较多。

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

图 12 官方 VGG-16 网络结构详解

4.2 图像的风格迁移

VGG 前面的卷积层从图像中提取“特征”，后面的全连接层把图片的“特征”转换为类别概率。其中，VGGNet 中的浅层（如 conv1_1，conv1_2），提取的特征往往是比较简单的（如检测点、线、亮度），VGGNet 中的深层（如 conv5_1，conv5_2），提取的特征往往比较复杂（如有无人脸或某种特定物体）[2]。

具体来说，风格迁移使用卷积层的中间特征还原出对应这种特征的原始图像。先选取一幅原始图像，经过 VGGNet 计算后得到各个卷积层的特征。接下来，根据这些卷积层的特征，还原出对应这种特征的原始图像在[2]。

4.3 代码逻辑

代码主要参考博客园[3]。

因为笔者能力问题，从零开始构建 VGG-16 网络还略困难，而且风格迁移需要大量图片训练，也没有相应的硬件条件，所以直接使用 pytorch 中的预训练模型 VGG-16。预训练模型由 ImageNet 数据训练得出。

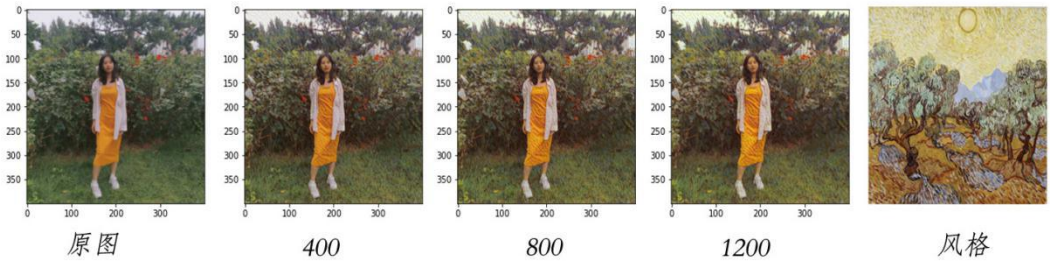
使用预训练好的 VGG-16 网络，使用十分方便，但在风格迁移网络中，我们需要获得中间层的输出，所以需要修改网络的前向传播过程，删除不需要的层，在代码中用 get_features 函数进行实现。

此外，一般使用 Gram 矩阵来表示图像的风格特征，注重风格纹理，而忽略空间。在代码中用函数 gram_matrix 进行实现。

定义两个损失，content loss 和 style loss，分别用来衡量生成图片与输入图片在内容、细节上的相似性和生成图片与风格图片在风格上的相似性。最终的 total_loss 用权重乘以 loss 的和来表示。

4.4 风格迁移结果

采用本人自拍分别用梵高和 Louis Icart 的画作作了风格迁移。随着迭代次数的不断增加，图片的变化如下。



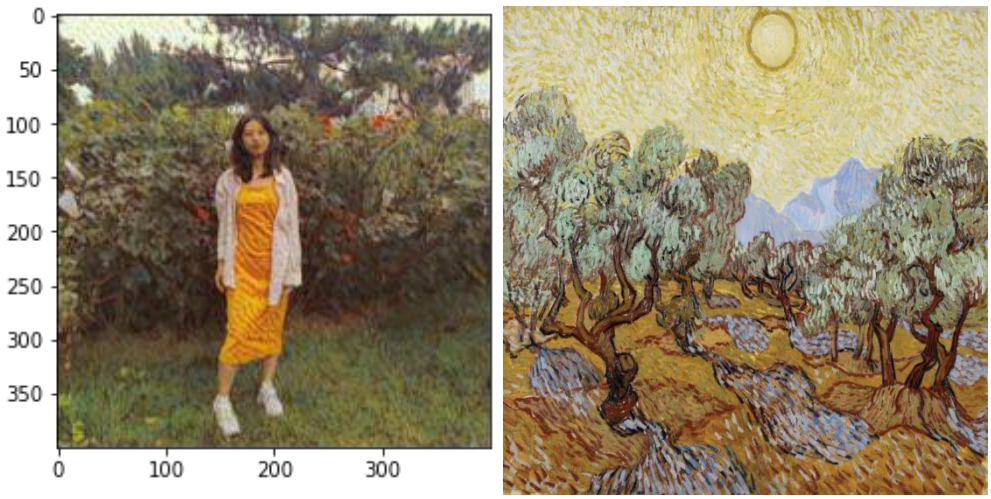
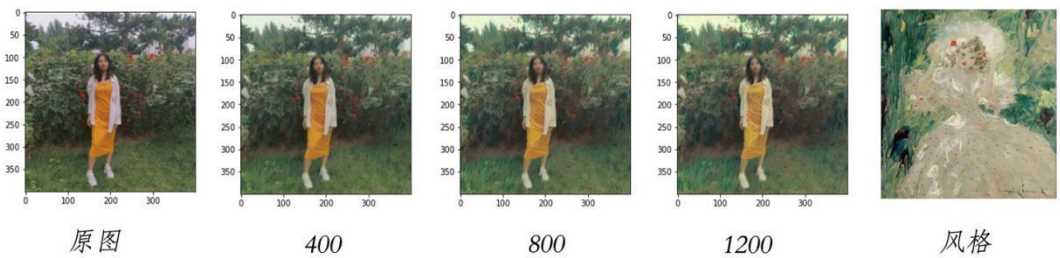


图 13 梵高风格迁移



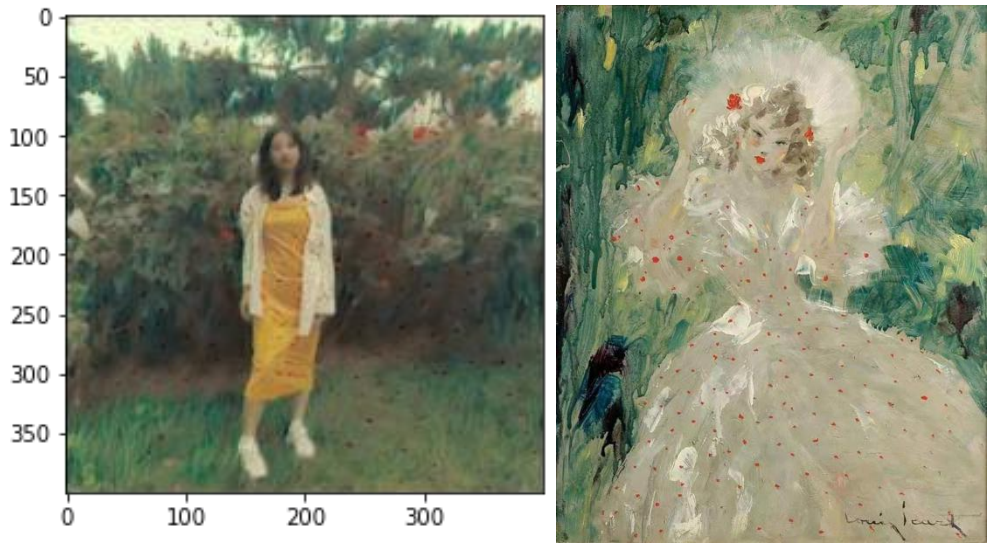


图 14 Icart 风格迁移

随着迭代次数不断增加，用 visdom 画出模型的 loss。

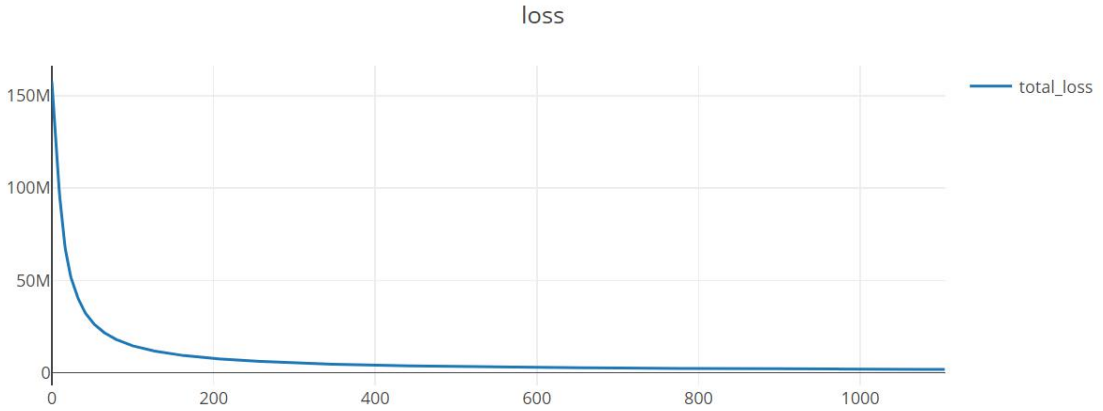


图 15 梵高风格迁移的 total_loss 变化

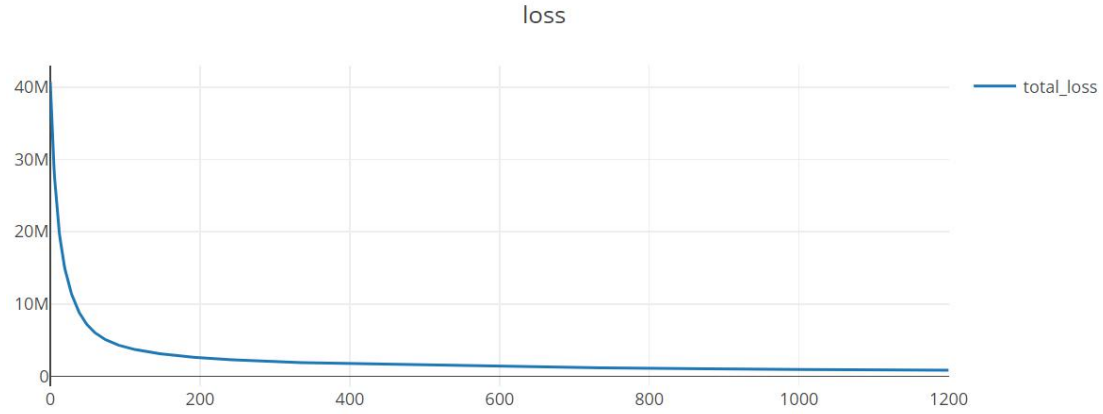


图 16 Icart 风格迁移的 total_loss 变化

5 总结

本文对卷积神经网络的探讨非常浅显，也没有什么创新点，有些基础部分可能有些啰嗦，但是是我本人学习理解的一个过程。在这个过程中，我也对卷积神经网络的结构、用法，以及如何

在 pytorch 中实现有了一个更深刻的理解，最重要的是，学习了如何使用 visdom 来将训练动态可视化。并尝试了对简单的卷积神经网络进行调参，来使模型的准确度提高。希望在以后的学习中，可以更加深入地去理解背后的数学知识。

参考文献

[1] 包俊,董亚超,刘宏哲. 卷积神经网络的发展综述[A]. 中国计算机用户协会网络应用分会. 中国计算机用户协会网络应用分会 2020 年第二十四届网络新技术与应用年会论文集[C].中国计算机用户协会网络应用分会:北京联合大学北京市信息服务工程重点实验室,2020:6.
[2] 博客园.https://www.cnblogs.com/xiaoyh/
[3] 博客园.https://www.cnblogs.com/MartinLwx/p/10572466.html